编译原理实验二报告

191870271 赵超懿

实现功能

1. 完成所有必做的语义分析及错误检查，能够识别17种要求的语义错误。

   完成所有选做的要求包括函数声明，变量受作用域影响，结构体结构等价。

2. 通过重新遍历语法分析树，进行自定向下分析，在每一个语法分析树的结点，添加语义动作，根据当前节点的类型，按照一定顺序计算其属性以及完成对符号表的操作，包括对符号表的检查，插入变量、类型定义、函数到符号表等操作。

3. 符号表使用哈希表，作用域的实现使用讲义提到的十字链表。

程序使用及编译

1. 在Code文件夹下运行make parser生成二进制文件parser
2. 运行命令./paser $FILE即可
3. make test做了一些修改，详细情况见makefile

程序特点及具体实现

项目结构

```
Code
 |---sytnax.y              //bison file
 |---lexical.l             //flex file
 |---data.c                //语法分析树
 |---semantic.c            //语义分析
 |---symbol.c              //符号表，栈等数据结构和一些基本操作
 |---data.h                //数据结构的定义
 |---debug.h               //debug
```

符号表的实现使用哈希表方式，作用域的实现使用讲义中提到的十字链表

接口设计

```
            Stack Begin --> 1 -->   2 --> ... --> End
Symbol Table                |       |
|pointer| --> List Begin -> Node -> Node -> ... -> List End
|pointer|                   |       |
|pointer| --> List Begin -> Node -> Node -> ... -> List End
|pointer|
```

符号表和栈以及节点采用这样的结构，构成十字链表

符号表和栈提供以下接口

```
typedef struct SymbolTable_t {
    ....
    int (*insert)(Symbol_Node_t *);         //插入节点
    int (*remove)(Symbol_Node_t *);         //删除节点
    Symbol_Node_t * (*find)(char *);        //查询元素，返回指针
    ...
}SymbolTable_t;

typedef struct SymbolStack_t {
    ...
    void (*push)(int);              //压栈
    void (*pop)();                  //出栈
    FieldList * (*pop_var)();       //返回当前作用域所有定义的变量
    SymbolStack_ele_t * (*top)();   //返回栈顶的元素
    ...
}SymbolStack_t;
```

语义分析的具体实现

对与每个节点实现一个完成语义动作的函数Semantic_Check_($Node_Type)

对每个节点根据语法调用对应的函数

以下一些问题的具体解决方案：

1. 函数声明和函数定义

思路是使得符号表中只有一个该函数的节点，通过改变该节点的类型（函数定义或函数声明）来判断函数是否已经定义。

当遇到函数定义时，检查符号表，查看是否已经有函数声明或定义：

若有定义，报错。若有声明，检查二者是否相符合，若符合，将符号表中的声明改成函数定义。

若没有，加入符号表。

2.DefList同时用于**struct** 和 局部定义的处理

使用SymbolStack中的pop_var接口，可以得到当前作用域所有的变量的列表